

Name :	Ms.Neerja Doshi
UID	2021300029
Div	SE CE DIV A – BATCH B
Experiment No. 6: (Graph Algorithm - Single source shortest path algorithm..Dijkstra Algorithms	

AIM :	(Graph Algorithm - Single source shortest path algorithm..Dijkstra Algorithms
THEORY :	<p>The Single-Source Shortest Path (SSSP) problem consists of finding the shortest paths between a given</p> <ol style="list-style-type: none"> 1. vertex v and all other vertices in the graph 2. Bellman Ford algorithm works by overestimating the length of the path from the starting vertex to all other vertices. 3. Then it iteratively relaxes those estimates by finding new paths that are shorter than the previously overestimated paths. 4. Dijkstra algorithm is a single-source shortest path algorithm. Here, single-source means that only one source is given, and we have to find the shortest path from the source to all the nodes. <p>5. ALGORITHM :</p> <p>1. Bellman–Ford :</p> <ol style="list-style-type: none"> 1. function bellmanFordAlgorithm(G, s) //G is the graph and s is the source vertex 2. for each vertex V in G 3. dist[V] <- infinite // dist is distance 4. prev[V] <- NULL // prev is previous 5. dist[s] <- 0 6. for each vertex V in G 7. for each edge (u,v) in G 8. temporaryDist <- dist[u] + edgeweight(u, v) 9. if temporaryDist < dist[v] 10. dist[v] <- temporaryDist 11. prev[v] <- u

	<ol style="list-style-type: none"> 12. for each edge (U,V) in G 13. If $\text{dist}[U] + \text{edgeweight}(U, V) < \text{dist}[V]$ 14. Error: Negative Cycle Exists 15. return $\text{dist}[], \text{previ}[]$ <p>2. Dijkstra Algorithms :</p> <ol style="list-style-type: none"> 1. Mark the source node with a current distance of 0 and the rest with infinity. 2. Set the non-visited node with the smallest current distance as the current node. 3. For each neighbor, N of the current node adds the current distance of the adjacent node with the weight of the edge connecting 0->1. If it is smaller than the current distance of Node, set it as the new current distance of N. 4. Mark the current node 1 as visited. 5. Go to step 2 if there are any nodes are unvisited. <p>7. Time Complexity :</p> <ol style="list-style-type: none"> 1. Bellman–Ford <ol style="list-style-type: none"> 1. $O(V * E)$ 2. Dijkstra Algorithms <ol style="list-style-type: none"> 1. $O((V+E)\text{Log}V)$
CODE :	<pre> #include <limits.h> #include <stdbool.h> #include <stdio.h> // Number of vertices in the graph #define V 9 // A utility function to find the vertex with minimum // distance value, from the set of vertices not yet included // in shortest path tree int minDistance(int dist[], bool sptSet[]) { // Initialize min value int min = INT_MAX, min_index; </pre>

```

        for (int v = 0; v < V; v++)
            if (sptSet[v] == false && dist[v] <= min)
                min = dist[v], min_index = v;

        return min_index;
    }

    // A utility function to print the constructed distance
    // array
    void printSolution(int dist[])
    {
        printf("Vertex \t\t Distance from Source\n");
        for (int i = 0; i < V; i++)
            printf("%d \t\t\t %d\n", i, dist[i]);
    }

    // Function that implements Dijkstra's single source
    // shortest path algorithm for a graph represented using
    // adjacency matrix representation
    void dijkstra(int graph[V][V], int src)
    {
        int dist[V]; // The output array. dist[i] will hold the
                    // shortest
        // distance from src to i

        bool sptSet[V]; // sptSet[i] will be true if vertex i is
                    // included in shortest
        // path tree or shortest distance from src to i is
        // finalized

        // Initialize all distances as INFINITE and stpSet[] as
        // false
        for (int i = 0; i < V; i++)
            dist[i] = INT_MAX, sptSet[i] = false;

        // Distance of source vertex from itself is always 0
        dist[src] = 0;

        // Find shortest path for all vertices
        for (int count = 0; count < V - 1; count++) {
            // Pick the minimum distance vertex from the set of
            // vertices not yet processed. u is always equal to
            // src in the first iteration.
            int u = minDistance(dist, sptSet);

            // Mark the picked vertex as processed
            sptSet[u] = true;

            // Update dist value of the adjacent vertices of the
            // picked vertex.
            for (int v = 0; v < V; v++)

```

```

        // Update dist[v] only if is not in sptSet,
        // there is an edge from u to v, and total
        // weight of path from src to v through u is
        // smaller than current value of dist[v]
        if (!sptSet[v] && graph[u][v]
            && dist[u] != INT_MAX
            && dist[u] + graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
    }

    // print the constructed distance array
    printSolution(dist);
}

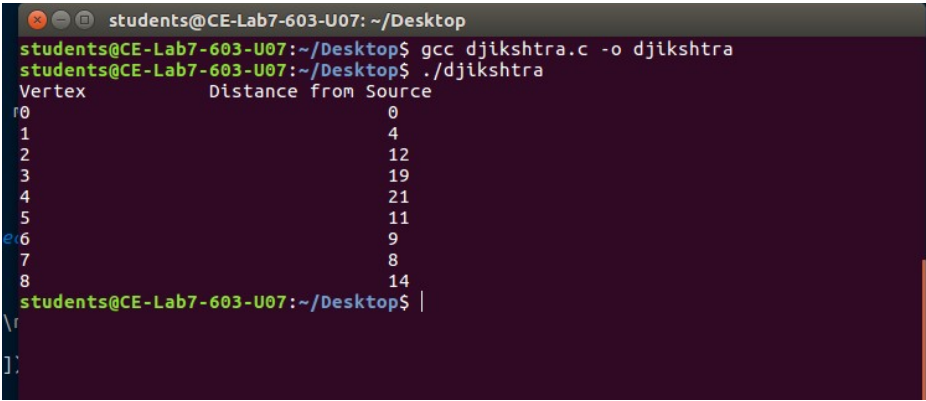
// driver's code
int main()
{
    /* Let us create the example graph discussed above */
    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                        { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                        { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

    // Function call
    dijkstra(graph, 0);

    return 0;
}

```

OUTPUT :



```
students@CE-Lab7-603-U07: ~/Desktop
students@CE-Lab7-603-U07:~/Desktop$ gcc djikshtra.c -o djikshtra
students@CE-Lab7-603-U07:~/Desktop$ ./djikshtra
Vertex          Distance from Source
0                0
1                4
2               12
3               19
4               21
5               11
6                9
7                8
8               14
students@CE-Lab7-603-U07:~/Desktop$ |
```

CONCLUSION : By performing the above experiment , i have successfully understood to perform